to the createInputMessage method previously described. The output message is labeled with the same message name as the input message except that the suffix added is "_REPLY." The constructor for Message class 44 creates a second DOM document within an instance of the DOM Document class at block 138. The second DOM document represents the output message.

At block 140, the Launch method calls a serviceMain method. The serviceMain method is an abstract method in BusinessService class 48, which gets overloaded by the subclass of BusinessService class 48 corresponding to the request and response. Overloading describes the condition where a method (serviceMain method) includes multiple implementations in different subclasses (the subclasses of BusinessService class 48), where each implementation provides functionality pertaining to the corresponding subclass.

When the serviceMain method gets overloaded, custom application code is executed at block 142. The custom application code is identified by the request name in the request. Execution of the custom application code associated with the subclass of BusinessService class 48 is directed by the serviceMain method. At block 144, the custom application code reads the XML structured first DOM document representing the input message and extracts the request parameters therefrom. The custom application code uses the request parameters in the first DOM document to extract data from the back-end systems layer 18 at block 146.

Referring now to FIG. 7, at block 148, the custom application code provides the extracted data to the business services layer 16 as a response to the input message. The createField method of Message class 44 is again executed at block 150 for each unit of data in the response. A plurality of versions of the createField method are available to simplify the conversion of the units of data in the response to the fields in the second DOM document representing the output message. Selection of the version of the createField method is based on the datatype of each unit of data. In one embodiment, the datatype of units of data may be short integer, long integer, Boolean, string or group. In other embodiments, greater or fewer datatypes may be included. Selection of the datatype is performed as part of developing the custom application code.

At block 152, the selected versions of the createField method create fields for the second DOM document. In addition, as a function of the version of the createField method, the corresponding text nodes are set to the unit of data. The fields are created to form the output message in similar fashion to the input message previously discussed. The selected versions of the setAttribute method of Field class 46 set the appropriate attributes for the newly created fields at block 154. Similar to the createField method, selection of the version of the setAttribute method is based on the datatype of each attribute value. In one embodiment, the datatype of attribute values may be short integer, long integer, Boolean, string or group. In other embodiments, greater or fewer datatypes may be included. The resulting output message is returned from the serviceMain method and, ultimately, from the Launch method.

Creation and Translation Of An Output Message To Desired Format

Once the Launch method completes, the doGet method of ApiService class 42 resumes control and initiates translation of the output message at block 156. Referring now to FIG. 8, at block 158, the request is checked to determine if the output message should be translated to XML-text format. If yes, a generateXML method within Message class 44 is executed at block 160. The generateXML method wraps an XML serializer to provide a textual representation of the virtual tree. At block 162 the second DOM document representing the output message is translated to XML-text by the generateXML method. The translated output message is returned to the end-user systems layer 12 via the front-end systems layer 14 at block 164.

If the output message should not be translated to XML-text format, the doGet method calls a precompileStylesheet method of Message class 44 at block 166. The precompileStylesheet method pre-compiles an XSL stylesheet used in translation from XML to the desired presentation format, such as, for example, HTML. At block 168, the doGet method executes a generatePresentation method of Message class 44. The generatePresentation method processes the output message with the pre-compiled XSL stylesheet to complete the translation to the desired presentation format at block 170. Both the precompileStylesheet method and the generatePresentation method wrap the XSL processor API and its associated methods. At block 164, the translated

output message is returned to the end-user systems layer 12 via the front-end systems layer 14.

FIG. 9 is an expanded block diagram of a portion of the e-commerce software architecture 10 of FIG. 2 depicting exemplary custom application code implemented in a subclass of BusinessService class 48. In the illustrated embodiment, BusinessService class 48 includes a first subclass that is DirLister class 172. The DirLister class 172 includes a first subclass that is a DirLister_Request class 174 and a second subclass that is a DirLister_Reply class 176. In addition, the back-end systems layer 18 includes a datafile 178 in operative communication with DirLister class 172 as illustrated. One embodiment of the custom application code for the DirLister class 172 is included in the computer program listing appendix filed herewith.

In the exemplary embodiment, DirLister class 172 is initiated by a request to read the contents of a directory within the datafile 178. In this example, the datafile is organized in a well-known data hierarchy which includes files contained in one or more directories and associated subdirectories. As a function of the request, DirLister class 172 may return data pertaining to the names, sizes, and modification dates of the files and subdirectories contained in the directory requested. Further, DirLister class 172 may also recursively retrieve similar data from files within the subdirectories associated with the directory requested. In this example embodiment, the amount of information returned for each file, as well as the traversal approach is selectable within the request.

Referring now to FIGs. 2 and 8, a user desiring information on a directory in the datafile 178 accesses the e-commerce software architecture 10 using delivery technologies within the end-user systems layer 12. A request with the request name "DirLister " is made via the front-end systems layer 14. The request includes request parameters indicating the directory of interest and selection of the amount of data desired. An exemplary request is:

request=DirLister&path=path name of directory*&short=true&long=false&
info=false&deep=false.

The request is translated by the business services layer 16 to an input message with the message name "DirLister_Request."